It's like flossing your teeth: On the Importance and Challenges of Reproducible Builds for Software Supply Chain Security

Marcel Fourné *, Dominik Wermke†, William Enck ‡, Sascha Fahl †, Yasemin Acar ¶
*Max Planck Institute for Security and Privacy, Bochum, Germany, marcel.fourne@mpi-sp.org
†CISPA Helmholtz Center for Information Security, Germany, [first.last]@cispa.de
‡North Carolina State University, Raleigh, North Carolina, USA, whenck@ncsu.edu
¶Paderborn University, Germany, George Washington University, USA, acar@gwu.edu

***Abstract*—The 2020 Solarwinds attack was a tipping point that caused a heightened awareness about the security of the software supply chain and in particular the large amount of trust placed in build systems. Reproducible Builds (R-Bs) provide a strong foundation to build defenses for arbitrary attacks against build systems by ensuring that given the same source code, build environment, and build instructions, bitwise-identical artifacts are created. Unfortunately, much of the software industry believes R-Bs are too far out of reach for most projects. The goal of this paper is to help identify a path for R-Bs to become a commonplace property.**

To this end, we conducted a series of 24 semi-structured expert interviews with participants from the Reproducible-Builds.org project, finding that self-effective work by highly motivated developers and collaborative communication with upstream projects are key contributors to R-Bs. We identified a range of motivations that can encourage open source developers to strive for R-Bs, including indicators of quality, security benefits, and more efficient caching of artifacts. We also identify experiences that help and hinder adoption, which often revolves around communication with upstream projects. We conclude with recommendations on how to better integrate R-Bs with the efforts of the open source and free software community.

1. Introduction

> "*To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.*" - Ken Thompson

Thompson's 1984 Turing award lecture "Trusting Trust" demonstrated that the security of a program is more than the logic in its source code [1]. It also includes all of the programs used to make the source code logic executable. Thirty-six years later, Thompson's theoretical attack became a pressing concern for nation states across the globe. The 2020 Solarwinds attack did not inject malicious source code into a project. It did not trick users into installing software with an invalid signature. The Solarwinds attack trojaned the *build system*, transparently injecting malicious logic into the product binary signed with Solarwinds' official code signing keys.

The world of software has transformed entirely since Thompson's lecture. His suggestion of "trusting the people" is orders of magnitude harder today than in 1984. Today's software supply chain ecosystem is vast, with software projects often including transitive dependencies that are tens to hundreds of layers deep. And despite much of the software supply chain being open source, end users rarely, if ever, compile software themselves. Thus, the build system has become an ideal target for attackers.

Reproducible Builds (often abbreviated R-Bs) offer a foundation for defending against attacks targeting the build system. "*A build is **reproducible** if given the same source code, build environment and build instructions, any party can recreate bit-by-bit identical copies of all specified artifacts*" [2]. Conceptually, R-Bs allow multiple parties to build the same software package, and assuming the attacker cannot simultaneously compromise the build systems of all parties, arbitrary subversion of an individual build system can be trivially detected. High-profile projects including Bitcoin [3] and Tor [4] use R-Bs to assure users that distributed binary executables match their source code. Recent literature has built upon R-Bs to enhance build provenance guarantees [5] and create verifiable builds [6] more suitable for providing guarantees to end customers. Finally, the US National Security Agency (NSA) identifies R-Bs as an important part of securing the software supply chain [7], [8].

Unfortunately, most software builds are not reproducible. Over the past decade, the Reproducible-Builds.org project has cataloged the many sources of non-determinism that prevent R-Bs, including uncontrolled build inputs (e.g., system time, environment variables, and build location) and build non-determinism (e.g., process scheduling) [9]. Despite industry [10], [11] and academic [12]–[14] tools to facilitate R-Bs and extremely high rates of R-Bs in some popular Linux distributions (95.5% for Debian on AMD64 [15] and 88.7% for Arch Linux core [16]), much of the software industry believes R-Bs to be a long-term goal, if achievable at all [17].

The goal of this paper is to help R-Bs more quickly become a commonplace effort in software development. We seek answers to the following research questions:

RQ1: "*What are motivations for, and common themes around, adopting reproducible builds in projects?*" We are interested in our participants' motivations around striving for reproducible builds in their projects, specifically in the case of complex, community- or industry-driven projects, that likely necessitate a complex, interconnected system of motivations and drivers. We are also interested if some of the motivations involve security, and what specific threat models are applied.
RQ2: "*What experiences and challenges did projects encounter in the context of reproducible builds?*" Most projects were not created with reproducibility in mind. We are interested in what experiences were made, and challenges encountered, on the way towards reproducibility, both by contributors of the project as well as with outside entities such as customers or upstream dependencies. This research question aims at the personal experiences of R-B developers.
RQ3: "*What are commonly encountered obstacles and facilitators in projects' efforts towards reproducibility?*" Some projects stall in their efforts toward reproducibility, while others succeed. We are interested in what facilitators and obstacles our participants encountered during their efforts, how they approached them, and what they would recommend for other projects aiming to become reproducible. This research question aims at external factors encountered by R-Bs developers.

By answering these questions, we hope to guide future industry and academic efforts that target both the technical and human aspects of R-Bs. In this paper, we report the results of a semi-structured interview study with 24 prominent and public members of the R-Bs effort. All participants were experienced developers (5+ years) with R-Bs experience, who could give deep insights into their thought and development processes, and deeply discuss and reflect on the topic. Based on these interviews, we offer the following key insights.

- *Open Source developers are self-motivated to work on software infrastructure.* They see themselves as users as well as developers and want to build better software even without external requests.
- *The Snowden revelations and SolarWinds incident heightened the security awareness.* While some people were interested in R-Bs before, their number grew significantly after those two public events.
- *Caching matters most to businesses.* R-Bs allow for efficient caching of artifacts, which was mentioned as the most important aspect for businesses.

While we specifically choose to interview experts with many invested years in R-Bs for their experience and insights, we also want to highlight that our expert participants are likely positively biased regarding the potential for R-Bs becoming widespread. While this may be substantiated by growing numbers of developers on the R-Bs mailing list as well as growing parts of operating systems being tested as built reproducibly, our sample is still biased towards R-B enthusiasts [18].

The remainder of this paper proceeds as follows. Section 2 discusses background information and related work. Section 3 describes our interview methodology. Section 4 presents our detailed results. Section 5 provides discussion and a set of recommendations. Section 6 concludes.

2. Background and Related Work

This section provides background information for reproducible builds and their relation to overall open source software supply chain security, as well as related work in three key areas: research on reproducible builds, open-source software security, and interview studies with software developers with a focus on software security.

Relationships between open source software projects can be characterized by their order of incorporating software: When one project uses another project's software, it is often called downstream of the project originating the software. Conversely, many project interactions are with upstream projects, which originated the software. A single package that does not have a downstream relationship with another package, instead of just redistributing it without changes, is called a leaf package. Any package that provides some functionality used in the infrastructure of a software build and supply tool chain can also be called an infrastructure package. To make a package build reproducibly, all of its dependencies need to build reproducibly. If a maintainer changes one package to do so, it is often most efficient to upstream those changes as a patch set for incorporation by the upstream developers, so the work is shared with all other downstream projects of that dependency.

2.1. Reproducible Builds Background Information

Reproducible builds are a collection of techniques and processes that aim to make the compilation of source to resulting binary code deterministic: the same source code should always be compiled to the bit-by-bit identical binary code. Achieving reproducibility in building software is non-trivial as the software compilation process is susceptible to multiple sources of non-determinism.

Lamb and Zacchiroli [19] provide an overview of common sources of non-determinism during the build process:[1]

Build timestamps are commonly used by C programming language projects and other build tools such as `make` [20] using similar macros to the `__DATE__` C-preprocessor macro. While build timestamps are useful for bug reporting, most version control systems including Git provides alternatives such as offer better solutions to record specific software versions without introducing additional non-determinism [21].

Build paths are commonly embedded using a C programming language preprocessor macro (e.g., `__FILE__`) as well as assertion macros referencing source code locations or log messages. In most cases, a relative path to the root location of the source code is sufficient [22] and reduces the amount of non-determinism. Build

1. This list is not meant to be exhaustive.

directory name inclusion can be prevented by only including build paths relative to the root directory of the source code, which will be constant. Non-constant paths can include user-specific changes. To avoid these while maintaining absolute build paths, a statically known directory path can be used for all software builds. This is used in Nix, GUIX, but also Debian using the `sbuild` software [23], which is the basis for the R-Bs checking tool `reprotest` [11].

Filesystem ordering as part of the POSIX standard does not define the order when returning the results of a directory listing causing additional non-determinism.

Archive metadata such as the date and ownership information in `.zip` and `.tar` archives are commonly used for archival purposes but should be avoided to reduce non-determinism.

Randomness results not only from parallelism and concurrency in the build process. Additionally, some compilers introduce explicit randomness during the build process to generate unique names that do not conflict with those generated for other files such as single compilation unit [24] distinct identifiers.

Unitialized memory adds non-determinism by not always initializing memory to a programmer-defined value and is supported by some popular low-level programming languages including C and C++.

As part of our interview study, we aim to better understand how open source software developers identify and handle these sources of non-determinism.

The reproducible-builds.org project [2] aims to support developers and software projects in making their build processes reproducible. They support mainstream Linux distributions such as Debian (currently 95.5% reproducible for AMD64) and Arch Linux (currently 88.7% reproducible for Arch Linux core) in building their software reproducibly.

The project provides several tools to help developers achieve R-Bs, including `reprotest` to automate the process of building a package multiple times in diverse environments and `diffoscope` [10] to help find the differences within complex binary packages and directories. `Diffoscope` receives two files to be compared as input, tries to unpack any data recursively, and displays differences found between the two input files. It is plugin-based and as a wrapper re-uses common file format handling utilities.[2]

As part of our work, we aim to learn if and how developers use support tools for R-Bs.

2.2. Research on Reproducible Builds

Most prior research considering reproducible builds addressed technical challenges. In 2005, Wheeler proposed diverse double compilation [25] to address Thompson's Trusting Trust attack [1], [26], connecting software security and R-Bs. Diverse double compilation suggests that source code is compiled by different compilers, potentially mutually distrusting parties, and every additional instance giving the same resulting binary file makes the Trusting Trust attack less likely. However, the software must build reproducibly, giving the exact same binary if compiled at different points in time, space, and for any number of recompilations. To generate a fully trustworthy compile chain, trustworthy root binaries are needed. A popular approach to address this issue is bootstrapable builds [27]: The core idea is to address circular build dependencies of complex software by creating a new dependency path using more simple prerequisite software.

Prior work suggested R-Bs as a primitive to improve software supply chain security including distributed verification [6], transparency logs [28], supply chain workflow integrations [5], and "keyless" signatures using trusted third-party inspectable logs [29].

Prior work has also considered better tool support for R-Bs. Several Linux distributions such as NixOS [30] and GNU GUIX [31]–[33] and their corresponding packaging tools are based upon primitives that promote reproducibility. In a series of work, Ren et al. proposed RepLoc [12], RepTrace [13] and RepFix [14] to more precisely locate sources of non-determinism and suggest patches. Recently, containers have been explored specifically for reproducibility [34] and corresponding security impact [35], as well as new build tools [36] which have been reimplemented as open-source tools. In closed- or mixed-source environments, rebuilds can be organized in a more centralized manner [37] that also supports R-Bs.

Closest to our work, Butler et al. interviewed company experts to investigate the value of R-Bs for companies [17]. They identified reasons for their limited adoption of R-Bs like limited awareness and perceived challenges.

In contrast to previous research on reproducible builds, our interview study aims to shed light on enablers and blockers for the adoption of R-Bs in the open source community.

2.3. Research on Open Source Software Security

Open source repositories are open to access from outsiders and the security and privacy research community has established use of this data source. From these repositories' commits [38]–[40] and contributors [41], but also secondary and related information like vulnerabilities [42], [43] and even torrents [44], [45] as an alternate access method have been used for research in a number of papers.

Big open-source projects like FreeBSD [46], Linux [47], and Mozilla [48] have been the focus of case studies. Topical analysis of vulnerabilities can take the corpus of code and has been done by matching Common Vulnerabilities and Exposures (CVEs) numbers [49], [50], by using the code base for evaluation of static analysis tools [51]–[54], or vulnerability changes [55]–[58]. These changes are necessary to secure a codebase, so the patterns and development of fixes have been investigated [59]–[61]. In one specific work, 337 CVE entries were linked to the patches fixing them and the authors found that the developers of those fixes are of

2. All provided tools can be found at https://reproducible-builds.org/tools/.

higher experience levels [62]. The highly polished Linux Kernel implementations of drivers have also been analyzed multiple times [63], [64].

The social aspects of repository contents researched things like toxic comments [65] and metadata [66], [67] as well as programming languages [68] and their general maintenance [69], [70]. Furthermore, pull requests [71]–[73], collaboration [74]–[76], and even gamification of the process [77] have been studied in related work.

In contrast to closed-source development, the security challenges for open-source communities are unique, valuable sources of research data and therefore well researched [78]–[80]. Open source software repositories contain public commits and issues that can be statistically evaluated [81], mined for emotions [82] and security tactics [83]. Programming language-specific communities were investigated, finding the Python and JavaScript communities to not react quickly to security vulnerabilities [84]. A large-scale analysis of hundreds of thousands code review requests from different open-source projects identified [85] the changes of less experienced contributors to be between 1.8 and 24 times more likely to contain security vulnerabilities.

Automated identification of open-source projects using vulnerability data [86]–[88] and toxic comments [89] were investigated. Both problems may weaken trust in the public reception of these software projects, even among collaborators. There has been work on different factors to influence [72], [90], [91] trust and quantification [92], [93] of it. Trust is influenced by open source projects' security, which itself is highly influenced by code quality, which has been investigated by different assessment models [94], the difference between architectural plans and implementation [95], and later code reviews [96], [97]. The base unit of collaboration is the committer, whose motivations [98]–[101], barriers to entry [102], [103], and the eventual pull requests [104] have been a focus of different works.

The onboarding [105]–[108] and mentoring [108], [109] of new committers have been studied as well. Socialization in the form of pre-existing relationships is an important factor and precursor [110] to joining GitHub projects. Most of such project ecosystems have one central project connecting every part of that ecosystem via software dependencies and connecting to other ecosystems as well [111].

In contrast to previous work on open-source software security, we focus on understanding enablers and blockers for the adoption of reproducible builds as a critical contribution to overall software security.

2.4. Interviews with Security Developers

The security and privacy research community uses interviews effectively and as a well-established method for detailed investigations. Prior work utilized interviews to gain detailed information about different kinds of experts, for example: Administrators [112], [113], and overall security professionals [114], [115], but also of communities that rely on their security, from journalists [116], over editors [117], to victim service providers [118]. Interviews can be used as a part of larger studies and give researchers a view of data that is not readily available from technical systems: e.g., thoughts and procedures. For Tor adoption [119] or how to work with encryption [120], examples are readily available, just like developers' thoughts and plans about security features [121].

In 2017, IT administrators were asked about the usability of deploying HTTPS [122], and programmers about the benefits and drawbacks of outright changing to a different programming language in a study in 2022 [123]. A 2021 qualitative study was performed about developers' struggles with CSP [124], and in 2022 industry practitioners' mental models of adversarial machine learning were investigated [125], [126].

Open-source developers are a special case, their transparency and distributed work being subject of dedicated study [74]. The social barrier of entry for new contributors was studied as part of a larger study containing semi-structured interviews with 36 developers recruited from 14 projects [102]. Their challenges and strategies for overcoming them using tasks recommended for newcomers were studied by interviewing mentors of 10 open-source projects [108]. Recently, Wermke et al. leveraged interviews to investigate behind-the-scene security processes in open source projects [127] and industry projects that utilize open source components [128].

As an extension to previous interview studies with security developers, we focus on reproducible builds.

3. Methodology

We conducted 24 in-depth, semi-structured interviews with developers, maintainers, and contributors implementing reproducible builds for their software in the fall of 2022. During these interviews, we discussed reasons for adopting R-Bs and the processes they encountered and used while doing this. Both the interview guide and the codebook are provided in the Appendix.

3.1. Participant Recruitment

We recruited participants by emailing 100 members of the Reproducible Builds Project website's mailing list "rb-general" [129]. We decided on this more focused recruitment approach, because based on our prior experiences we assumed that developers from outside the mailing list would often not be familiar with the concept of R-Bs. Some participants made additional suggestions and/or offered introductions to potential interviewees with insights into reproducible builds; we followed up on all suggestions.

We interviewed a total of 24 participants between August and November 2022. Table 1 provides an interviewee demographics summary. All of our participants are software developers (5+ years) with reproducible builds experience. They provided us with insights into their thoughts on R-Bs and development processes, and could discuss and reflect on the topic. Five participants stated that they are engaging less with R-Bs than at some point in the past, never fully joined the R-B effort, or have not yet started actively working

towards R-Bs, while still being interested, to hear their reasons against R-Bs. The set of developers we interviewed contained two non-binary individuals and otherwise men, which is disappointing for diversity, but reflects those active in the mailing list and community we targeted.

3.2. Interview Procedure

We conducted semi-structured interviews and topical hints to keep the interviews flowing and otherwise let our participants structure their answers and remarks on their own following established practices [18]. We built our interview guide around our research questions, and discussed and revised it with researchers outside our team with reproducible builds experience. Figure 1 illustrates the interview structure and we provide our final interview in the Appendix.

To pre-test the interview guide, we conducted one mock interview. Interviews generally lasted between 30 to 60 minutes. We scheduled the first batch of interviews with the intention of treating them as pilot-interviews, however, as no major changes were made to the interview guide and the data we collected from them was meaningful, we decided to include theses pilot interviews in our data set. Throughout the interviews, we asked for experiences and opinions and participants' responses indicate that they also reported their (strong) opinions in either direction of R-Bs, including reasons against R-Bs. We specifically iterated over the (various) definition(s) for reproducibility and discussed them with our interviewees.

We offered our interviewees to choose between a locally hosted Jitsi and Zoom for the remote interviews and conducted six of the interviews in person at the Reproducible Builds Summit 2022.[3] We gave them the option to end the interview and withdraw at any time. We started interviews with verbal consent to being audio-recorded, the interview being transcribed by a GDPR-compliant third-party service and the use of the interviews in a scientific publication. One or two authors conducted and recorded the interviews.

1. Context and Definition. The interview guide opened with a section establishing the context for participants, their projects, and their role in the projects. Questions included how, when, and why they got in touch with their projects, their background, and how they decided to focus on reproducible builds. In addition, we established their definition of a reproducible build for their project and in general.

2. Reasons and Decisions. The *"Reasons and Decisions"* section explored the reasons for progressing towards reproducible builds in the project and specific decisions surrounding this process. Questions included internal and external drivers for pursuing reproducible builds, threat models and requirements, and how decisions were formed and by whom.

3. Processes and Tools. The *"Processes and Tools"* section established utilized processes and tools for the reproducible builds, as well as experiences with these approaches. Questions included the general process of making the project or parts of it reproducible, the estimated time for these efforts, how the experiences with upstream projects were, and whether additional resources like documentation or websites were any help.

3. https://reproducible-builds.org/events/venice2022/

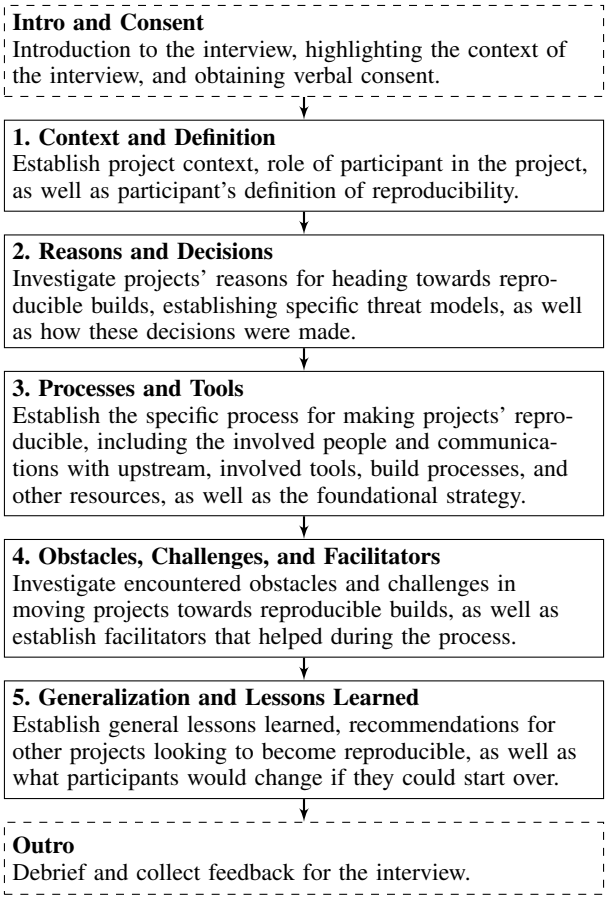

Figure 1. Illustration of topic flow in the reproducible builds interviews. As we conducted semi-structured interviews, participants were presented with general questions and corresponding follow-ups in each section, but were generally free to diverge from this flow.

4. Obstacles, Challenges, and Facilitators. The *"Obstacles, Challenges, and Facilitators"* section investigated encountered obstacles, challenges, and positive facilitators during the projects' progress towards reproducible builds. Questions included obstacles of different types (organizational, technical, dependencies / upstream, and communities) and which particular factors were helpful for the project.

5. Generalization and Lessons Learned. The interview closed with a *"Generalization and Lessons Learned"* section, establishing general themes and lessons learned from their reproducible builds attempts. Questions included what they would do differently if they could start over, what worked well and what did not, and what they would recommend for other developers and projects attempting reproducible builds.

After the interview, participants often expressed their interest in our research and in being mailed results (which we

promised) and/or recommended other potential interviewees. The authors debriefed with each other after the interviews, discussing new insights.

3.3. Reproducible Builds Summit Discussion

We presented preliminary results from this study, mostly on what motivates reproducible builds and how they can benefit stakeholders, to attendees at the Reproducible Builds Summit 2022, Venice. Attendees, together with the main author, then used a collaborative session to iterate over a matrix of motivations and benefits for reproducible builds, which we (with the attendees' consent) discuss with insights into motivations and experiences from our interviews in Section 5.

3.4. Coding and Analysis

After the 18th interview, new themes ceased to emerge and participants mostly iterated themes we had heard before; therefore, we chose to stop interviewing after 24 participants, reaching theoretical saturation [130]. We analyzed our data using qualitative coding [18]. The main author developed a codebook based on the interview guide and insights from conducting the interviews; the team reviewed and slightly iterated over the codebook. The main author then coded all data with the codebook, discussing insights with the team. The final codebook is provided in the Appendix.

3.5. Ethical Considerations and Data Protection

Our study, including recruitment strategy, data collection, recording methods, and interview guide, was approved by our Institutional Review Board (IRB).

All participants were either recommended by their own colleagues or signed up to a public list of those generally interested in reproducible builds and reacted accordingly to our invitation emails: They generally responded positively, expressed interest in our research, recommended others, and attempted to schedule interviews. We offered online (and offline) meeting and recording options, including meeting them at the Reproducible Builds Summit that an author attended, as well as using self-hosted meeting software (Jitsi) with local recordings for improved privacy. All participants consented to the interviews, recordings, and transcription by an external, GDPR-compliant service. When participants flagged parts of the interview as too sensitive to transcribe, we removed them before transcription. We de-identified participants and interview transcripts using identifiers such as P01 and de-identified sensitive information in the transcripts. After checking transcripts for correctness, we deleted all audio recordings. We offered no compensation, due to our interviewees being potentially highly paid individuals, motivated to work on R-Bs by their involvement in the Open Source/Free Software project communities; based on our prior experience with this population, they usually reject payments or attempt to re-direct them to donations for OSS projects, which our funding source was unable to accomplish.

3.6. Limitations

Our work is affected by limitations common to interview studies, including limited generalizability and biases such as recall and social desirability biases [18]. We accounted for these biases by interviewing a diverse (in projects and experience) sample of those who fit our recruitment criteria. Furthermore, we only interviewed those involved in the Reproducible Builds project; therefore, our sample represented the experiences and perceptions of those who were generally highly aware of and/or working with reproducible builds. While interviewees happened to be concentrated in the Western world and were predominantly male, this is in line with the reproducible builds community [131]. Our sample includes various organizational contexts, from industry-leading companies to single developers. Based on provided answers, we can reason that our sample is broad and diverse in R-B adoption, experiences, and organizational contexts, but we refrain from comparing smaller and larger companies quantitatively due the limited sample sizes and the more qualitative nature or our research approach.

4. Results

In this section, we describe the findings from 24 semi-structured interviews with developers with experience in R-Bs for software projects. First, we illustrate our interviewees' motivation to implement R-Bs for their software projects. Second, we explore supporting factors and obstacles for R-Bs. We de-identified participant quotes, made minor grammatical corrections, and highlighted omissions using brackets ("[...]"). German interview quotes were translated into English by native German speakers.

Counts in our reporting should be interpreted as the number of interviewees that touched on the specific topic at least once during their interview. As qualitative interview study, reported counts are not necessarily representative for the wider developer population, but are included to give some general idea about the distribution of codes and to highlight especially prevalent or underrepresented themes in the interviews.

Table 1 provides an overview of project demographics. We conducted 18 remote interviews and six in-person interviews at the Reproducible Builds Summit 2022. Four of those interviewees were recruited at the summit. We mark those interviews "PS" (vs. "P" for all other participants), as their attitude towards R-Bs might be particularly positive. Most (21) interviewees worked in some capacity as developers on projects that strived to build reproducibly. All interviewees were software developers with between 5 and more than 20 years of experience in general software development, specifically between 2 and 12 years on R-Bs.

TABLE 1. OVERVIEW OF OUR INTERVIEW PARTICIPANTS

Alias	Interview			Project		
	Duration	Codes[1]	Recruitment Channel	Position	Area	Software Stack[2]
P01	44 minutes	40	rb-general mailing list	Developer	Operating systems	Ocaml
P02	31 minutes	15	rb-general mailing list	Developer	Desktop Environments	C
P03	42 minutes	29	rb-general mailing list	Developer	Operating systems	C/C++
P04	39 minutes	27	rb-general mailing list	Developer	Operating systems	Assembly, C, and others
P05	39 minutes	31	rb-general mailing list	Developer	Operating systems	C, Tcl
P06	51 minutes	40	rb-general mailing list	Developer	Operating systems	diverse
P07	45 minutes	29	rb-general mailing list	Developer	Operating systems	C, Python
P08	58 minutes	14	rb-general mailing list	Developer	Graphics processing	C/C++
P09	50 minutes	28	rb-general mailing list	Developer	Operating systems	Python, a little C/C++
P10	55 minutes	36	rb-general mailing list	Developer	Operating systems	C
P11	57 minutes	24	rb-general mailing list	Developer	Privacy preservation	C/C++, Rust
P12	54 minutes	20	rb-general mailing list	Developer	Build systems, GUIs	Python, C and others
P13	64 minutes	19	Personal recommendation	Developer	Operating systems	C and others
P14	50 minutes	15	rb-general mailing list	Project lead	Electronic currencies	diverse
P15	34 minutes	22	rb-general mailing list	Advisor	Privacy infrastructures	-
P16	42 minutes	26	Personal recommendation	CEO	Build systems	Python and others
P17	39 minutes	20	rb-general mailing list	Developer	Operating systems	C/C++, Python, Scheme, and others
PS18[3]	45 minutes	24	RB Summit 2022, Venice	Developer	Embedded software	C, Assembly, and others
P19	31 minutes	11	RB Summit 2022, Venice	Developer	Privacy preservation	C/C++, Rust, and others
PS20[3]	48 minutes	27	RB Summit 2022, Venice	Developer	Operating systems	Scheme, C, and others
P21	49 minutes	34	RB Summit 2022, Venice	Developer	Operating systems	diverse
PS22[3]	46 minutes	24	RB Summit 2022, Venice	Developer	Operating systems	diverse
PS23[3]	58 minutes	29	RB Summit 2022, Venice	Developer	Build systems	Java
P24	31 minutes	35	rb-general mailing list	Developer	Operating systems	C/C++ and others

[1] Total number of codes assigned to the interview after resolving conflicts.
[2] Abbreviated. Common among all participants was some amount of shell script use.
[3] Participant aliases: P means participant was recruited by email, PS indicates recruitment at Reproducible Builds Summit.

4.1. Why and How Projects Started to Work on Reproducible Builds

In this section, we illustrate reasons for and against reproducible builds that the interviewees mentioned.

Reasons for and against adopting Reproducible Builds. We identified technical and non-technical themes related to our participants' motivations for making their builds reproducible, both related to security and as an intuition of how compilation should behave according to their own mental model of software compilation.

Ten participants reported encountering a misunderstanding of the mechanics of current software compilers: They brought up that they commonly encounter the expectation that compilers generally produce the same binary output given the same source code without outside interference.

> "*I have an input and some computation, so I expect the output to be the same. [Like a mathematical function.] And like a scientific function. It's computations; you put something in it, and the same output should get out. Except if the function randomizes, [...] or it is broken. I think unreproducible builds are illogical. Conversely, reproducible builds are logical.*" - P21

Their main motivation to work on reproducible builds was to bring the mechanics of software compilation closer to their assumptions. Aside from compiler mechanics, broken expectations were brought up by eleven interviewees. Beside their expectation for compilation working deterministically, they also want software to work the same in the future. Two participants reported beneficial (better run-time performance in some cases or security fixes) but still unexpected compiler behavior:

> "*It's more like things aren't fixed. You do a deployment one day with some source code, and you come back a week later, and you do the deployment again. You repeat the process with the same source code. Your source code is the same, but because you haven't engineered the process to be reproducible, what you actually deploy is something different. [...] [T]oo often, one, it's not understood what's changing and two, you don't have control over it.*" - PS22

Some interviewees (6) mentioned the importance of constantly maintaining a high level of build quality to limit increased effort later in the development process. For 18 interviewees, improved software quality was the main motivator. One participant compared the effects of improving software quality by making it build reproducibly to dental flossing:

> "*At every summit you have people show up there because they want the hands-on support to get their thing into a more reproducible status so [...] I have this ongoing analogy I use around dental flossing and the dentists tell you how important it is to the dental floss and people do not very often floss as much as their dentist tells them to. I think Reproducibility is falling in that same spectrum of*

really important things [and] people treat it like that." - P15

A similar sentiment was reported for company resources:

"*The only reason why we ever moved into this direction of reproducible builds, for the company, was because it was causing us issues in losing time. People would forget to declare dependencies and that would fail the build. People merge small changes that change the dependency ordering when executed massively parallel. We noticed that when this happened, it would take us half a day to fix. During this half of the day, there were about 500 people who couldn't build anymore. That costs a lot of money and time.*" - P05

As described earlier, many participants started working on R-Bs due to intrinsic motivation; they began with the parts of their software projects that best fit their motivation. Once they had worked on their chosen package, they explored more complex components required for R-Bs in their software packaging work.

The main motivation for 16 interviewees was working on infrastructure reproducibility, while six worked on single software packages. While building infrastructure solves more problems compared to leaf packages, individual configuration specific R-Bs problems are specific to leaf packages.[4] Overall, participants reported complex, interconnected motivations to implement R-Bs; some motivations are related to intuition for how builds should behave, and some are grounded in explicit security concerns, both for developers and software and its users. One developer mentioned working on *version pinning*. Version pinning is an approach of describing the exact version of each software dependency, as an easy, but necessary area to implement R-Bs by documenting a set of dependency versions that produced a working artifact. Another two interviewees mentioned that they worked on difficulties with specific compiler versions, citing specific versions of well-known compilers as problematic for R-Bs. Older versions of those compilers generated more reproducible binaries than current versions. Finally, two participants mentioned to have worked on *transitive dependencies*, i.e., dependencies of dependencies, which may influence reproducibility. These participants reported having investigated if indirect dependencies broke their reproducibility and figured out how to fix this.

Four participants mentioned interactions with upstream projects. While they had made a version of their software that depended on upstream projects being reproducible, they needed the upstream project to incorporate and maintain their changes. One interviewee reported that they had an upstream project rewrite a suggested patch from scratch, and were amazed at the commitment to R-Bs by the upstream project. The ideal goal for all participants was full bit-by-bit reproducibility. However, projects have considered weaker reproducibility criteria as more realistically achievable along the way.

4. See https://github.com/bmwiedemann/theunreproduciblepackage.git for a list of known problems that can occur in a leaf package.

Two participants started with the build process by manually debugging unreproducibility introduced by it. After achieving reproducibility manually, 15 participants continued their efforts toward R-Bs through more automation in Continuous Integration (CI) and other infrastructure.

With R-Bs, a build of the same source code version results in the exact same binary. Without R-Bs, a rebuild can change the binary of the package, which can still be cached each time but leads to waste and incompatibilities. Ten developers, across industry and open source projects, mentioned that slow build speeds are frustrating for developers, and can be caused by inefficient caching that is sensitive to small changes. Building reproducibly would solve this problem.

"*We recently got a new build machine which is I think 64 cores and [lots] of memory or something like that, terabytes of this as a benchmark. How long does it take to build everything from scratch? It took about a day on that machine to build everything. [...] Fortunately, that's not the typical dev experience because our particular system caches build outputs.*" - P11

The caching strategy mentioned by P11 only applies, if the compilation toolchain works like a mathematical function, giving the exact same result to an unchanged input, i.e., if compilation builds reproducibly.

Reasons against working on Reproducible Builds. Thirteen interviewees also mentioned reasons against R-Bs. They reported that, due to decreasing enthusiasm and repetitive work, they decreased the amount of time and effort they invested in R-Bs. This decrease in enthusiasm and effort corresponded with a perceived impracticality of fully reproducible builds due to workload, missing organizational buy-in, unhelpful communication with upstream projects, or the goal being perceived as only theoretically achievable. Relatedly, the frustrating experience of "moving goalposts" was described as follows: Projects that had achieved bit-by-bit R-Bs might "lose" that status when someone found a previously unchecked source of mutation in the environment that broke full reproducibility.

Seven participants mentioned discussions about which mutations of the build environment should be checked for fully reproducible software packages, including a feeling of unclear goals, as noted by P01:

"*[Projects] also have various definitions of what reproducibility means. It means you have some mysterious cloud, and in the end, you get the very same binaries, the bit-wise identical binary. That is the output, but what is considered as part of the input is not very clear.*" - P01

Seven participants, again across industry and OSS, mentioned detractors to implementing R-Bs, such as missing organizational buy-in, as mentioned by P10:

"*To say, 'Reproducibility is stupid; go away.' That happens very rarely. We just remember it a lot because it's interesting. The most common inter-*

action with upstream is silence. They just don't merge the patches," - P10

or due to their unwillingness to change their build process, as mentioned by P06: "*[project] is an old project and some areas are very conservative.*" (P06) One interviewee compared a lack of reaction to suggestions that contribute to R-Bs to a general quality problem in projects:

> "*[E]arly warnings that you know this and that upstream has some problems with their definition or they don't want to accept the patch about certain epoch, specification, or something like that.*" - P07

The notion of unclear goals that change over time when a new source of unreproducible behavior occurs, was echoed by P08:

> "*Because we are taking a 'fix when we find' approach, I don't think we are doing anything to evaluate whether a package has achieved 100% reproducibility. Usually, when we fail to register a package to Reprepro, we know there may be a reproducibility issue and will look into it. Once we fix it, we'll do a clean build a few times to make sure the same binaries are produced. It sounds more like 'we make sure it's reproducible for now'.*" - P08

Personal reproducibility target changes were discussed in 13 interviews. These changes in goals include being content with repeatable builds (i.e., compiling the same source code at different times into a functional artifact, not necessarily with the same bit-by-bit result) instead of fully R-Bs. One participant expressed hope that the community would value and work on the guarantees fully R-Bs provide over e.g., repeatable builds:

> "*It's one of those things where if more people in the community value this, they would get solved. Right now, they're happy with the builds being reproducible-ish.*" - P04

Summary: Reasons for and against adopting Reproducible Builds. Interviewees were mostly driven by improved software quality. The unclear impact of R-Bs on the overall security of deployed software systems, together with high effort, were reasons against R-Bs.

Reproducible Builds as a Protection Mechanism. By design, R-Bs can serve as a protection mechanism for software projects against security and quality problems that might be introduced—maliciously, through coercion, or through mistakes—by software developers. Many interviewees (14) mentioned R-Bs as a security measure against coercive attacks against developers by malicious actors:

> "*The time where you have a room full of Debian developers and you tell them, 'If your computer is compromised in a way, then you might unknowingly compromise millions of machines'. People were like, 'I am this kind of target.' [...] It was a way to remove some leverage for a malicious actor to actually go at the people directly. If you would try to kidnap my kids and say, 'You need to plant this malware.' I can say, 'I can't. It's going to be seen, so probably you should do it differently and give me back my kids.' If you don't have reproducible build systems in place, then they have leverage because it's going to go unnoticed if you release a binary that doesn't match the actual source code.*" - P06

R-Bs were also mentioned as a requirement for checking OSS, and making explicit trust in individual maintainers redundant. Some participants (13) mentioned that explicit trust in open source software project maintainers was not needed since open source code can be audited by anyone, at any given time. This possibility of having one's work audited was discussed as an incentive for honesty, and not wanting to be publicly seen as dishonest, which in turn was discussed as a powerful motivator motivator to safeguard the security of open-source software projects against powerful (non-)government agencies and private security actors. R-Bs were discussed as one strong mechanism to facilitate public scrutiny. In turn, participants discussed that for R-Bs to be an effective public security measure, openness is a requirement.

Eleven participants mentioned that their user base requested improved security, including specific requests for R-Bs, based on an awareness of one or more highly impactful, publicized security incidents, including the Snowden leaks, the Heartbleed vulnerability, the introduction of the GDPR, or the SolarWinds incident.

> "*It was clearly a need. There were a lot of the NSA, Snowden revelations coming out, and various things like that. Nothing specifically from those revelations, but the gist of a general vibe of, oh, we can trust even less than we previously thought as a very general idea, and the world, post-2015, is moving to a more data-conscious and privacy security.*" - P10

Although many participants mentioned requests for reproducibility due to highly visible security incidents, they also explained that these requests originate from only a small part of their user base—security-affine power-users—that reported concerns and requested R-Bs as a preventative measure against software supply chain attacks.

> "*It is perceived as one of those very rarely visited corners and only in cases like breaches and things like SolarWinds and attacks against suppliers.*" - P07

Most of our interviewees (23) could not name a security incident related to reproducible builds, either thwarted or caused by it. However, one interviewee mentioned the following case:

> "*I had a package that was not reproducible when I checked. The difference between my rebuilds and the reference builds that are in [binary software repository] is the passphrase of a GPG key. During the builds, it was recorded because the command line that he used for signing, he added it in the parameter. I don't know why he's recorded*

it into a file that was then merged into the archive, and it is in [binary software repository]." - PS23

In the above example, R-Bs helped to detect a secret leak and contributed to the overall security of the project.

> Summary: Reproducible Builds as a Protection Mechanism. Attacks on software supply chains were an important driver to build software reproducibly.

Software Project Decision Structures. For both, open source and commercial software projects, interviewees reported that decision structures had a strong impact on R-Bs.

Although many OSS projects make important decisions by consensus, only seven participants reported a joint decision process to start moving the project to implement R-Bs. Some interviewees (11) reported that individual developers in their projects had independently started work on R-Bs due to intrinsic motivation.

For commercial projects, decisions were driven by commercial product deadlines, and decisions were generally made at the management level. Five participants reported that their project started moving towards implementing R-Bs through management decisions, heavily influenced by developers' insistence on what they perceived to be a contribution towards product quality.

> Summary: Project Decision Process. Individual developers have an influence on R-Bs adoption. They either drove R-Bs adoption by starting the process themselves or influencing project decisions.

4.2. Experienced Obstacles

In this section we report obstacles our interviewees experienced while working on R-Bs. Obstacles include unexpectedly large efforts, unsupportive upstream projects, and development processes that might be unusable, undefined, chaotic, or inconsistent. We discuss challenges and opportunities relating to community support in Section 4.3.

Lack of Good Communication. In general, eleven interviewees mentioned the relevance of strong communication skills to implement R-Bs in open source software. This communication involved heavy discussions of the concept of R-Bs, the goals achieved by R-Bs, and the need to adopt R-Bs. A common theme related to communication was a lack of outreach. The Reproducible Builds project's outreach consists of maintaining a website and mailing list, as well as paying one developer to post monthly progress reports, as they report on reproducible-builds.org. This included building the website's infrastructure for reporting successes in R-Bs, including testing OSS projects for reproducibility criteria themselves. However, to move beyond outreach and towards a wide adoption of R-Bs, participants discussed the need for wider support beyond those already involved in the R-Bs project. Participants communicated that more people working on R-Bs would be beneficial and would move the needle on software supply chain security. Eleven participants reported feeling not having done enough outreach-related work themselves, having instead focused on solving individual reproducibility changes in the code for which they felt responsible. One interviewee discussed the issue of transferring research advances into code, a problem also seen in prior research [132]:

"*I think there's this big gap between scientific research and programming. [. . .] I think bootstrappability and reproducible builds could be an enormous boost for free software, and inspire people to move towards free software and free software practices.*" - P17

Eight interviewees mentioned a lack of and a higher need for more helpful documentation for R-B efforts in a software project. The documentation on the R-Bs central website was also mentioned as needing more work.

Seven of our interviewees mentioned experiences of unhelpful interactions with other developers or users who were unsupportive or had difficulties understanding the concepts and benefits of R-Bs and the required effort. This led to delays in achieving R-Bs, including projects for which the upstream communication is still ongoing or stuck in a bug tracker. A total of eleven developers were astonished by the amount and importance of good communication for R-Bs. Eleven interviewees mentioned patience as a virtue in communicating with upstream projects that were less motivated to make reproducible builds part of their project. Participants explained that "good etiquette" in the open-source ecosystem is to proliferate bug reports upstream, ideally accompanied by a patch. The goal behind this approach is to fix issues at the source, getting rid of the burden of maintenance of the local changes. Our interviewees often went beyond that: five told us that they iterated with upstream projects and "polished" their patches until the upstream projects accepted them.

> Summary: Lack of Good Communication. The impact of interaction with other developers was often initially underestimated; participants discussed the importance of patience and good communication.

Technical Obstacles. Binaries including dates or other point-of-time information were the most common issues for reproducible builds reported by our interviewees. Reasons our interviewees gave included debugging artifacts, and different software version commits. eight interviewees mentioned the `SOURCE_DATE_EPOCH` standard [133] as an effective solution to the above problems. The `SOURCE_DATE_EPOCH` standard replaces random build time information with epoch.[5]

Five interviewees mentioned build directory name inclusion (cf. Section 2) that hindered their adoption of R-Bs.

"*There has been some pitch to also support build path prefix, someway, somehow, but I don't know how to use it. From my approach, the OCaml compiler is not relocatable at the moment. It will be in the future eventually, but I'm not too concerned about it because I don't think there's any threat model that contains the build path, and in the*

5. The epoch value is the UNIX system time 01 Jan 1970 00:00:00 UTC

end, I'm fine with recording the operating system packages, and the environment variables that led to that binary. Then I'm conducting the builds in a container, or in jail." - P01

Three interviewees mentioned that compilers include randomness explicitly during the compilation process, but also that a deterministic initial value can be supplied (for example via `gcc -frandom-seed-string`):

"*I think we set the python seed. I think we set the python seed with Python. Um. The sort of state I'm trying to think of the other languages that they have, other weird things like that.*" - PS20

Only one interviewee mentioned the potential issues around Profile-Guided Optimization [134], which change optimizations based on execution on the compiling machine. four interviewees reported (embedded) cryptographic signatures as implemented in the Apple ecosystem with enforced cryptographic signatures on binaries as a problem for R-Bs. Seven interviewees mentioned an unclear definition of build reproducibility as an obstacle to adopting R-Bs. Discussions in the community range from full, bit-by-bit R-Bs down to repeatable builds.

Overall, most interviewees mentioned that the technical obstacles above are manageable for people interested in R-Bs, but there is a long tail of problems to fully R-Bs.

> Summary: Technical Obstacles. Interviewees reported a wide range of technical obstacles including embedded timestamps, signatures, and build directories, which they assessed as intellectually simple, but cumbersome and repetitive to solve.

4.3. Helpful Factors

Helpful factors most mentioned were being self-effective, which interviewees defined as being determined, possessing the skill-set to progress R-Bs, and having good communication with other developers.

Self-effective Participants. 14 interviewees implemented R-Bs by themselves, through trial and error with some software that they tried to reproduce, which they described as the most efficient pathway to R-Bs. They described that a self-effective work environment, including having ownership of a large part of a project, being able to implement changes themselves, and prioritizing tasks as well as work packages themselves greatly contributed to effective work on R-Bs. We did not discover any different path to R-Bs in our interviews. Relatedly, interviewees explained hardships in increasing community efforts towards R-Bs, as the reserve of developers productively contributing to R-Bs. Anyone who could meaningfully contribute would need a high level of specialized knowledge and familiarity with the project; however, working on R-Bs might not be the most attractive work that these highly skilled open source developers might want to work towards.

> Summary: Self-effective Participants. Interviewees reported that specialized knowledge about projects, as well as the enthusiasm and ability to take broad action made their R-Bs work possible.

> These circumstances seems hard to scale to volunteers from the broader community.

Successful Community Communication. We identified factors that support R-Bs, often centering around effective community interaction. 14 participants mentioned positive interactions with upstream projects. These interactions included benign disinterest:

"*Well, the first thing I do is ask them if they're aware of the problem because there's still a decent percentage of even developers that aren't fully aware of the reproducibility problem. I think it's solved at the package manager level. You know, they think that there's some sort of layer of it and it gets kind of resolved. Then from there, you know, I help them to understand the types of things that like this scope can help them understand the ways to ability can be compromised.*" - P15

Positive interactions also included encouraging cooperation and work on upstreamed patches: "*We got in contact with upstream and they fixed it, and now it's working fine.*" (P19) In addition to interacting with upstream projects, communication with compiler authors was mentioned as helpful. Six interviewees reported positive interactions with compiler authors regarding R-Bs. Like other upstream requests, interviewees reported providing patches to compiler authors to address R-Bs issues, which were later incorporated.

> Summary: Successful Community Communication. Being helpful to upstream projects helps create goodwill for R-Bs in the open source community. Upstream projects may spend time and effort on R-Bs if they receive help from the R-Bs community for their own software project.

Helpful Resources. Different resources that helped with R-Bs were discussed in the interviews, but the most helpful resource for implementing R-Bs can be summarized as "good tooling." A total of 13 people mentioned that they found tooling particularly helpful, specifically the `diffoscope` tool [10]. Eight interviewees mentioned that projects should work on the future seamless integration of R-Bs into the build process. Eight interviewees mentioned additional resources they used for R-Bs, including the R-B's website and mailing list. Eight interviewees stated that documentation should be expanded to make onboarding new R-Bs enthusiasts easier. In contrast, P16 stated that existing documentation was sufficiently helpful for R-Bs, and that efforts should increase community awareness and buy-in toward more effective support for R-Bs.

5. Discussion

In the previous results sections, we establish the importance of effectively disseminating the benefits of R-Bs to a broader community and that communication with upstream projects is crucial.

Outreach. To better communicate the procedural, monetary, reputational, and general benefits that stakeholders can gain

from employing reproducible builds, we discussed preliminary findings and potential benefits with participants of the Reproducible Builds Summit 2022 in Venice with additional related material provided in the Appendix, and highlight key points from that discussion below.

The goal of an outreach effort is to describe benefits that a stakeholder of fully reproducibly built software may want. The benefits can be classified as time savings, monetary savings, reputational gains, and generally better results of the work with the software. A non-exhaustive list of stakeholders includes non-university research groups, universities, development corporations, security organizations (which can themselves be in a corporation), open-source projects, end users, and governmental organizations. For any of the potential beneficiaries of reproducible builds, multiple benefits may apply. While any user of the software may be an end user individually, most organizations have different needs as a whole.

Almost any software project can benefit from caching build results, giving decreased build times and better turnaround times for changes. A published reproducible build will not change and can be reproduced exactly as-is, so no retesting is needed since all previous tests for that build directly apply for a rebuild. Build debugging is simpler since any singular build that fails for some users can be reproduced for finding the bugs in it. Developers can work a lot faster and with more confidence that bugs they introduce can be bisected and figured out, while at worst, any previous version can be used. The open-source project can also save on hardware resources, since build artifacts can be deduplicated effectively, meaning that only parts that changed need to be saved again. This can also be used for updates, where only the differences between updated versions need to be transferred to each user, saving bandwidth (cost). For their reputation, the open source project can fulfill more parts of the OpenSSF scorecard [135]. While the project does not directly gain a reputation from higher software quality in its dependencies, however, choice of dependencies has an effect on the reputation of the open-source project in question. Incorporating reproducibly built dependencies may therefore indirectly increase a project's reputation.

In general, open source projects gain faster builds and the ability as well as a guarantee that they can build their software at any point in the future. All of this applies with one additional benefit: They learn which software components were used in building their software. This binary introspection can be a hard task in itself, but R-Bs can give this information almost for free using "`.buildinfo`"-files or a Software Bill of Materials (SBOM), as elaborated in analogy by P21:

> "*I don't grow plants, I don't create food, I don't write recipes, I don't prepare meals, but I do research on how to tell people to wash their hands. This does not make the food taste any better, make it better in general, more healthy. Maybe it makes the food a little bit less unhealthy or poisonous, but most times you'll be fine eating unwashed food. We try to change the way food is prepared. This does not influence the food at first but makes it better, safer, more healthy in general. We want to change the way people prepare food, just like water sewage, and treatment systems have been installed before. We want to change the mindset. We want to remind people, that reproducible, deterministic software is possible and reasonable.*" - P21.

Answers to RQs. Our 24 semi-structured interviews with experts involved with reproducible builds projects provided the following answers to our research questions:

RQ1: "*What are motivations for, and common themes around, adopting reproducible builds in software projects?*"

Our interviewees mentioned complex, interconnected motivations in the context of R-Bs. Some motivations are related to intuition for how builds should behave such as being deterministic as well as motivations grounded in explicit security concerns such as a compromised maintainer account.

RQ2: "*What experiences and challenges did projects encounter in the context of reproducible builds?*"

Many interviewees mentioned positive interactions with upstream projects and other developers, although some specifically noted that upstream communication required patience.

RQ3: "*What are encountered obstacles and facilitators in projects' efforts towards reproducibility?*"

Commonly encountered obstacles to reproducible builds include build directory name inclusion and cryptographic signatures on the technical side, as well as patience and good social communication on the interaction side.

Additional Insights. Some interviewees suggested that the overall awareness and buy-in for R-Bs was lacking, and that even with the increase of prevalence of software supply chain attacks, R-Bs is not yet widespread. Participants reported that in the early days of the R-Bs effort, most work appears to have been invested into infrastructure, including the upstreaming patches and creation of tooling, which should in theory provide a foundation for developers to make leaf packages build reproducibly.

Based on our participants' answers there appears to be a still ongoing public discussion about which criteria need to be fulfilled to call a package reproducible. The clearest criterion is bit-by-bit identical build results, which we opted to use in this paper. However, even bit-by-bit identical builds is subjective with respect to the mutators used to evaluate packages. Currently, the most used deployment of `reprotest` does not check all available mutators while testing a package for its build reproducibility.[6] A test with those or other not yet found sources of unreproducible behavior may change some bits in the artifact and give way to a stricter definition. Definitions of R-Bs that allow for differences in files from embedded signatures make R-Bs on

6. https://reproducible.debian.net/, which now points to https://reproducible-builds.org/citests/ and is used by different OSS distributions

Apple devices possible. These definitions specify elements in files that may be otherwise bit-by-bit reproducible.

Many groups do not enforce R-Bs. The Debian policy suggests rather than enforces reproducibility, which is understandable: users want to use software even if it is not built in a reproducible way. The OpenSSF scorecard also only cites R-Bs inside a high-risk criterion named "non-reviewable code," a detail that is fairly buried in the documentation.[7] In the case of open source projects, missing organizational buy-in comes in the form lack of support to mark R-Bs bug reports as blocking for a new software release. Untested changes from R-Bs can break build systems, so projects being conservative about patches is understandable.

For transitive dependency problems, concrete technical documentation could be achieved by the pervasive use of Software Bills of Materials (SBOMs) [136] to indicate all software included in building an artifact, so the transitive dependencies could be traced over a dependency graph.

Neither of President Biden's Executive Orders for Cybersecurity [137] or Supply Chains [138] mentions R-Bs. While this promises some eyes on the criticality of (Software) Supply Chain Security, we are worried that neither adequate funding for the work of mostly hobbyist open source developers nor real changes or competent help are to come in the foreseeable future. Lack of funding and organizational buy-in for their work was one of the major detractors mentioned in our interviews.

While we do not have insights into governmental regulation efforts, at some point we expect to see some regulation about software, similar to regulations about mandatory fitness of purpose and non-toxicity for other products. As also mentioned in one of our interviews, there are legislative efforts underway towards requiring an SBOM, which can only be reliably generated by having the depth of information as is needed for R-Bs.

Recommendations. A significant effort by a small number of individuals has laid the groundwork for R-Bs, fixing hard-to-find non-determinism in common build infrastructure. Despite these efforts, a significant number of participants regretted not spending more time on outreach. The knowledge and frameworks around R-Bs have reached a level of maturity such that now is the time for broader consumption. In this light, we conclude with the following recommendations.

1. We urge the industry to give their engineers leeway to work on what they deem necessary for software quality. They were the ones hired for their expertise and to know what is necessary for this and they should be empowered to work on it. Missing organizational buy-in by supporting their developers who may already want to get rid of some technical debt was one of the main detractors for R-Bs. Industry funding and engineering freedom were specifically mentioned in our interviews as wished for items regarding buy-in.
2. The open-source community should join the R-Bs effort and make it the new standard, so new releases are reproducible by default. Newly released unreproducible software should be permissible in distributions only if sufficient reasons and a plan to change are provided, creating more security for their users and themselves. Help with upstream interaction, R-Bs developers' small numbers, fatigue, and clarifying the status and importance of packages on the last mile to 100% R-Bs could help a lot. R-B's goal was seen as not clearly communicated, which led to some burnout with part of our interviewees. Better communication and avoiding "moving goalposts" would minimize the reported loss of emotional investment.
3. Not based on our interviews, but rather related work on the Trusting Trust attack and Software Supply Chain Security, we see R-Bs as a potentially greater interest to the security research community. We hope more buy-in from security organizations and researchers could be achieved by treating unreproducible software builds as a serious threat for software supply chain security and better support reproducible builds. We see some similarity of the R-Bs effort and security concerns, hinted at by one participant's remark about the search for mutators that make builds unreproducible, just like security vulnerabilities are searched for.
4. Governments should mandate some level of R-Bs as part of a general effort to strengthen software quality. Liability for last-level commercial, for-profit entities should be a necessary precondition for being allowed to profit from software products, just like it is common with physical products. This would create financial incentives for companies to provide R-Bs as a part of their software quality and security.

7. https://github.com/ossf/scorecard/blob/main/docs/checks.md

6. Conclusion

While R-Bs offer a strong foundation for securing the software supply chain, much of the industry believes it is out of reach. We conducted a series of 24 semi-structured expert interviews with participants from the ReproducibleBuilds.org project with the goal of identifying insights that could lead to R-Bs becoming more commonplace in software development. Our findings include that the collaboration between highly motivated developers and upstream projects over long periods of time is a key aspect for the success of R-Bs. We identified a range of motivations for adopting R-Bs (**RQ1**), including indicators of quality, security benefits, and more efficient caching of artifacts. Discussions around process (**RQ2**) and obstacles (**RQ3**) confirmed many of the challenges discussed in this work.

The R-Bs effort to date has operated under the mindset of "infrastructure before leaf packages." It has required active and self-guided bug hunting to root out problems in the build infrastructure that have been long overlooked. While this approach has brought R-Bs far with very limited resources and persons, progress was in most cases achieved

with only limited organizational buy-in, specifically by motivated individual open-source developers.

When companies *do* care for R-Bs, it is mostly seen as a cost-saving measure and only sometimes as a safeguard against wasting the time of highly-paid software engineers. The goals of quality and robust security motivate open source developers a lot more than corporate developers, though this may change due to the current geopolitical climate. In particular, new US initiatives have made R-Bs very tacitly, but mostly indirectly named, important to (inter-)national security. However, while the software supply chain security is seeing generous amounts of funding, nothing yet has been earmarked towards R-Bs.

Acknowledgments

This work is supported in part by NSF grants CNS-2206865 and CNS-2207008. Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the view of funding agencies. We want to thank all interviewees for their participation and appreciate the valuable time that they have generously given. We also want to thank the anonymous reviewers for their valuable feedback.

References

[1] K. Thompson, "Reflections on trusting trust," *Commun. ACM*, pp. 761–763, 1984.

[2] Reproducible Builds project, https://reproducible-builds.org/docs/definition/.

[3] devrandom, *Gitian: A secure software distribution method*, https://github.com/devrandom/gitian-builder, 2011.

[4] M. Perry, S. Schoen, and H. Steiner, *Reproducible builds moving beyond single points of failure for software distribution*, https://media.ccc.de/v/31c3_-_6240_-_en_-_saal_g_-_201412271400_-_reproducible_builds_-_mike_perry_-_seth_schoen_-_hans_steiner, 2014.

[5] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "In-toto: Providing farm-to-table guarantees for bits and bytes," in *Proceedings of the 28th USENIX Security Symposium (Sec'19)*, Aug. 2019, pp. 1393–1410.

[6] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, *et al.*, "CHAINIAC: Proactive Software-Update transparency via collectively signed skipchains and verified builds," in *Proceedings of the 26th USENIX Security Symposium (Sec'17)*, Aug. 2017, pp. 1271–1287.

[7] *ESF Partners, NSA, and CISA Release Software Supply Chain Guidance for Suppliers*, https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3204427/esf-partners-nsa-and-cisa-release-software-supply-chain-guidance-for-suppliers/, 2022.

[8] *NSA, CISA, ODNI Release Software Supply Chain Guidance for Developers*, https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3146465/nsa-cisa-odni-release-software-supply-chain-guidance-for-developers/, 2022.

[9] C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2022.

[10] *Diffoscope in-depth comparison of files, archives, and directories.* https://diffoscope.org/, 2014.

[11] *Reprotest*, https://salsa.debian.org/reproducible-builds/reprotest, 2016.

[12] Z. Ren, H. Jiang, J. Xuan, and Z. Yang, *Automated localization for unreproducible builds*, 2018.

[13] Z. Ren, C. Liu, X. Xiao, H. Jiang, and T. Xie, "Root cause localization for unreproducible builds via causality analysis over system call tracing," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 527–538.

[14] Z. Ren, S. Sun, J. Xuan, X. Li, Z. Zhou, and H. Jiang, "Automated patching for unreproducible builds," in *Proceedings of the 44th ACM International Conference on Software Engineering (ICSE'22)*, 2022, pp. 200–211.

[15] *Reproducible Debian overview*, https://tests.reproducible-builds.org/debian/reproducible.html, 2023.

[16] *Reproducible Arch Linux?!* https://tests.reproducible-builds.org/archlinux/archlinux.html, 2023.

[17] S. Butler, J. Gamalielsson, B. Lundell, *et al.*, "On business adoption and use of reproducible builds for open and closed source software," *Software Quality Journal*, Nov. 2022.

[18] J. Lazar, J. H. Feng, and H. Hochheiser, *Research methods in human-computer interaction*. Morgan Kaufmann, 2017.

[19] C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," *IEEE Software*, vol. 39, no. 2, pp. 62–70, Mar. 2022.

[20] T. J. Thompson, "Designer's workbench: Providing a production environment," *The Bell System Technical Journal*, vol. 59, no. 9, pp. 1811–1825, 1980.

[21] https://git-scm.com/book/en/v2/Git-Basics-Tagging.

[22] https://android.googlesource.com/platform/build/soong/+/master/docs/best_practices.md.

[23] *sbuild*, https://salsa.debian.org/debian/sbuild.

[24] S. R. Schach, *Practical Software Engineering*. 1992.

[25] D. A. Wheeler, "Fully countering trusting trust through diverse double-compiling," *CoRR*, vol. abs/1004.5534, 2010.

[26] D. A. Wheeler, "Countering trusting trust through diverse double-compiling," in *Proceedings of the 21st IEEE Annual Computer Security Applications Conference (ACSAC'05)*, 2005.

[27] https://bootstrappable.org/.

[28] M. Linderud, "Reproducible builds: Break a log, good things come in trees," 2019, Master Thesis.

[29] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS'22)*, 2022, pp. 2353–2367.

[30] E. Dolstra, A. Löh, and N. Pierron, "NixOS: A purely functional Linux distribution," *Journal of Functional Programming*, vol. 20, no. 5-6, pp. 577–615, 2010.

[31] L. Courtès, "Functional Package Management with Guix," in *European Lisp Symposium*, Madrid, Spain, Jun. 2013.

[32] L. Courtès and R. Wurmus, "Reproducible and User-Controlled Software Environments in HPC with Guix," in *2nd International Workshop on Reproducibility in Parallel Computing (RepPar)*, Vienne, Austria, Aug. 2015.

[33] L. Courtès, "Code staging in GNU Guix," in *Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE'17)*, 2017, pp. 41–48.

[34] O. S. Navarro Leija, K. Shiptoski, R. G. Scott, *et al.*, "Reproducible containers," in *Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*, 2020, pp. 167–182.

[35] A. Zerouali, T. Mens, G. Robles, and J. Gonzalez-Barahona, *On the relation between outdated docker containers, severity vulnerabilities and bugs*, 2018.

[36] M. Ivanković, G. Petrović, R. Just, and G. Fraser, "Code coverage at Google," in *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'19)*, 2019, pp. 955–963.

[37] R. Potvin and J. Levenberg, "Why Google stores billions of lines of code in a single repository," *Commun. ACM*, vol. 59, no. 7, pp. 78–87, Jun. 2016.

[38] A. Pietri, D. Spinellis, and S. Zacchiroli, "The software heritage graph dataset: Public software development under one roof," in *Proceedings of the 16th International Conference on Mining Software Repositories (MSR'19)*, 2019, pp. 138–142.

[39] ——, "The software heritage graph dataset: Large-scale analysis of public software development history," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR'20)*, 2020, pp. 1–5.

[40] A. Alali, H. Kagdi, and J. I. Maletic, "What's a typical commit? a characterization of open source software repositories," in *Proceedings of the 16th IEEE International Conference on Program Comprehension*, 2008, pp. 182–191.

[41] G. Robles, L. Arjona Reina, A. Serebrenik, B. Vasilescu, and J. M. González-Barahona, "FLOSS 2013: A survey dataset about free software contributors: Challenges for curating, sharing, and combining," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR'14)*, 2014, pp. 396–399.

[42] A. Gkortzis, D. Mitropoulos, and D. Spinellis, "VulinOSS: A dataset of security vulnerabilities in open-source systems," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18, Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 18–21.

[43] M. Shahzad, M. Z. Shafiq, and A. X. Liu, "A large scale exploratory analysis of software vulnerability life cycles," in *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, 2012, pp. 771–781.

[44] G. Gousios and D. Spinellis, "GHTorrent: GitHub's data from a firehose," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR'12)*, 2012, pp. 12–21.

[45] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean GHTorrent: GitHub data on demand," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR'14)*, 2014, pp. 384–387.

[46] T. T. Dinh-Trong and J. M. Bieman, "The FreeBSD project: A replication case study of open source development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 481–494, 2005.

[47] Q. Tu *et al.*, "Evolution in open source software: A case study," in *Proceedings of the 2000 International Conference on Software Maintenance*, IEEE, 2000, pp. 131–142.

[48] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.

[49] N. Edwards and L. Chen, "An historical examination of open source releases and their vulnerabilities," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS'12)*, 2012, pp. 183–194.

[50] A. D. Householder, J. Chrabaszcz, T. Novelly, D. Warren, and J. M. Spring, "Historical analysis of exploit availability timelines," in *Proceedings of the 13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, 2020.

[51] K. Altinkemer, J. Rees, and S. Sridhar, "Vulnerabilities and patches of open source software: An empirical study," *Journal of Information System Security*, vol. 4, no. 2, pp. 3–25, 2008.

[52] M. Alenezi and Y. Javed, "Open source web application security: A static analysis approach," in *2016 International Conference on Engineering & MIS (ICEMIS)*, 2016, pp. 1–5.

[53] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta, "How open source projects use static code analysis tools in continuous integration pipelines," in *Proceedings of the 14th IEEE International Conference on Mining Software Repositories (MSR'17)*, 2017, pp. 334–344.

[54] M. Zahedi, M. Ali Babar, and C. Treude, "An empirical study of security issues posted in open source projects," in *Proceedings of the 51st Hawaii International Conference on System Sciences (HICSS'18)*, 2018, pp. 5504–5513.

[55] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni, "When are OSS developers more likely to introduce vulnerable code changes? a case study," in *Open Source Software: Mobile Open Source Technologies*, Springer Berlin Heidelberg, 2014, pp. 234–236.

[56] P. Anbalagan and M. Vouk, "Towards a unifying approach in understanding security problems," in *Proceedings 20th International Symposium on Software Reliability Engineering (ISSRE'09*, 2009, pp. 136–145.

[57] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai, "Bug characteristics in open source software," *Empirical software engineering*, vol. 19, no. 6, pp. 1665–1705, 2014.

[58] J. Walden, "The impact of a major security event on an open source project: The case of OpenSSL," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR'20)*, 2020, pp. 409–419.

[59] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, May 2005.

[60] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*, 2017, pp. 2201–2215.

[61] R. Ramsauer, L. Bulwahn, D. Lohmann, and W. Mauerer, "The sound of silence: Mining security vulnerabilities from secret integration channels in open-source projects," in *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, ser. CCSW'20, Virtual Event, USA: Association for Computing Machinery, 2020.

[62] V. Piantadosi, S. Scalabrino, and R. Oliveto, "Fixing of security vulnerabilities in open source projects: A case study of Apache HTTP server and Apache Tomcat," in *Proceedings of the 12th IEEE Conference on Software Testing, Validation and Verification (ICST'19)*, 2019, pp. 68–78.

[63] P. Deligiannis, A. F. Donaldson, and Z. Rakamaric, "Fast and precise symbolic analysis of concurrency bugs in device drivers," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*, 2015, pp. 166–177.

[64] J.-J. Bai, J. Lawall, Q.-L. Chen, and S.-M. Hu, "Effective static analysis of concurrency use-after-free bugs in Linux device drivers," in *Proceedings of the 2019 USENIX Annual Technical Conference (ATC'19)*, 2019, pp. 255–268.

[65] C. Miller, S. Cohen, B. Vasilescu, and C. Kästner, ""Did You Miss My Comment or What?" understanding toxicity in open source discussions," in *Proceedings of the 44th International Conference on Software Engineering (ICSE'22)*, 2022.

[66] D. Sondhi, A. Gupta, S. Purandare, A. Rana, D. Kaushal, and R. Purandare, "Dataset to study indirectly dependent documentation in GitHub repositories," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 215–216.

[67] R. Li, P. Pandurangan, H. Frluckaj, and L. Dabbish, "Code of conduct conversations in open source software projects on Github," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, no. CSCW1, Apr. 2021.

[68] W. Li, N. Meng, L. Li, and H. Cai, "Understanding language selection in multi-language software projects on GitHub," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 256–257.

[69] H. Hata, R. G. Kula, T. Ishio, and C. Treude, "Research artifact: The potential of meta-maintenance on GitHub," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 192–193.

[70] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, Paderborn, Germany: Association for Computing Machinery, 2017.

[71] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, 2014, pp. 345–355.

[72] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in GitHub," in *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, 2014, pp. 356–366.

[73] D. Ford, M. Behroozi, A. Serebrenik, and C. Parnin, "Beyond the code itself: How programmers really look at pull requests," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS'19)*, 2019, pp. 51–60.

[74] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: Transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW'12)*, 2012, pp. 1277–1286.

[75] B. Vasilescu, K. Blincoe, Q. Xuan, *et al.*, "The sky is not the limit: Multitasking across GitHub projects," in *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*, 2016, pp. 994–1005.

[76] K. Constantino, M. Souza, S. Zhou, E. Figueiredo, and C. Kästner, "Perceptions of open-source software developers on collaborations: An interview and survey study," *Journal of Software: Evolution and Process*, e2393, 2021.

[77] L. Moldon, M. Strohmaier, and J. Wachs, "How gamification affects software developers: Cautionary evidence from a natural experiment on github," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE'21)*, 2021, pp. 549–561.

[78] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, *Understanding free/open source software development processes*, 2006.

[79] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/libre open-source software development: What we know and what we do not know," *ACM Comput. Surv.*, vol. 44, no. 2, 2008.

[80] S.-F. Wen, "Software security in open source development: A systematic literature review," in *2017 21st Conference of Open Innovations Association (FRUCT)*, 2017, pp. 364–373.

[81] L. P. Hattori and M. Lanza, "On the nature of commits," in *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops*, 2008, pp. 63–71.

[82] D. Pletea, B. Vasilescu, and A. Serebrenik, "Security and emotion: Sentiment analysis of security discussions on GitHub," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR'14)*, 2014, pp. 348–351.

[83] J. C. S. Santos, A. Peruma, M. Mirakhorli, M. Galstery, J. V. Vidal, and A. Sejfia, "Understanding software vulnerabilities related to architectural security tactics: An empirical investigation of Chromium, PHP and Thunderbird," in *Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA'17)*, 2017, pp. 69–78.

[84] G. Antal, M. Keleti, and P. Hegedűns, "Exploring the security awareness of the Python and JavaScript open source communities," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR'20)*, 2020, pp. 16–20.

[85] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni, "Identifying the characteristics of vulnerable code changes: An empirical study," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 257–268.

[86] H. Perl, S. Dechand, M. Smith, *et al.*, "VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*, 2015, pp. 426–437.

[87] I. Abunadi and M. Alenezi, "Towards cross project vulnerability prediction in open source web applications," in *Proceedings of the The International Conference on Engineering & MIS 2015*, ser. ICEMIS '15, Istanbul, Turkey: Association for Computing Machinery, 2015.

[88] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 11th ACM Joint Meeting on Foundations of Software Engineering (ESEC/FSE'17)*, 2017, pp. 914–919.

[89] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. Vasilescu, "Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions," in *Proceedings of the ACM/IEEE 42nd Interna-*

tional Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER'20), 2020, pp. 57–60.

[90] M. Antikainen, T. Aaltonen, and J. Väisänen, "The role of trust in OSS communities — case Linux kernel community," in *Open Source Development, Adoption and Innovation*, J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, Eds., Boston, MA: Springer US, 2007, pp. 223–228.

[91] V. S. Sinha, S. Mani, and S. Sinha, "Entering the circle of trust: Developer initiation as committers in open-source projects," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11, Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 133–142.

[92] S. Bugiel, L. V. Davi, and S. Schulz, "Scalable trust establishment with software reputation," in *Proceedings of the sixth ACM workshop on Scalable trusted computing*, 2011, pp. 15–24.

[93] M. Syeed, J. Lindman, and I. Hammouda, "Measuring perceived trust in open source software communities," in *Open Source Systems: Towards Robust Practices*, F. Balaguer, R. Di Cosmo, A. Garrido, F. Kon, G. Robles, and S. Zacchiroli, Eds., Cham: Springer International Publishing, 2017.

[94] A.-K. Groven, K. Haaland, R. Glott, and A. Tannenberg, "Security measurements within the framework of quality assessment models for free/libre open source software," in *Proceedings of the 4th European Conference on Software Architecture (ECSA'10): Companion Volume*, 2010, pp. 229–235.

[95] J. Ryoo, B. Malone, P. A. Laplante, and P. Anand, "The use of security tactics in open source software projects," *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1195–1204, 2016.

[96] A. Bosu and J. C. Carver, "Impact of developer reputation on code review outcomes in OSS projects: An empirical investigation," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'14)*, 2014.

[97] C. Thompson and D. Wagner, "A large-scale study of modern code review and security in open source projects," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, Toronto, Canada: Association for Computing Machinery, 2017.

[98] A. Hars and S. Ou, "Working for free? motivations for participating in open-source projects," *Int. J. Electron. Commerce*, vol. 6, no. 3, pp. 25–39, Apr. 2002.

[99] C. Hannebauer and V. Gruhn, "Motivation of newcomers to FLOSS projects," in *Proceedings of the 12th International Symposium on Open Collaboration (OpenSym'16)*, 2016.

[100] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER'16)*, 2016, pp. 112–123.

[101] C. Miller, D. G. Widder, C. Kästner, and B. Vasilescu, "Why do people give up FLOSSing? a study of contributor disengagement in open source," in *Open Source Systems*, Springer International Publishing, 2019, pp. 116–129.

[102] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social barriers faced by newcomers placing their first contribution in open source software projects," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (SCW'15)*, 2015, pp. 1379–1392.

[103] S.-F. Wen, "Learning secure programming in open source software communities: A socio-technical view," in *Proceedings of the 6th International Conference on Information and Education Technology*, ser. ICIET '18, Osaka, Japan: Association for Computing Machinery, 2018.

[104] V. N. Subramanian, I. Rehman, M. Nagappan, and R. G. Kula, "Analyzing first contributions on GitHub: What do newcomers do," *IEEE Software*, 2020.

[105] I. Steinmacher, T. U. Conte, C. Treude, and M. A. Gerosa, "Overcoming open source project entry barriers with a portal for newcomers," in *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*, 2016, pp. 273–284.

[106] I. Steinmacher, C. Treude, and M. A. Gerosa, "Let me in: Guidelines for the successful onboarding of newcomers to open source projects," *IEEE Software*, vol. 36, no. 4, pp. 41–49, 2019.

[107] J. Dominic, J. Houser, I. Steinmacher, C. Ritter, and P. Rodeghero, "Conversational bot for newcomers onboarding to open source projects," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20)*, 2020, pp. 46–50.

[108] S. Balali, U. Annamalai, H. S. Padala, *et al.*, "Recommending tasks to newcomers in OSS projects: How do mentors handle it?" In *Proceedings of the 16th International Symposium on Open Collaboration (OpenSym'20)*, 2020.

[109] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "Who is going to mentor newcomers in open source projects?" In *Proceedings of the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE'12)*, 2012.

[110] C. Casalnuovo, B. Vasilescu, P. Devanbu, and V. Filkov, "Developer onboarding in GitHub: The role of prior social links and language experience," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'15)*, 2015, pp. 817–828.

[111] K. Blincoe, F. Harrison, and D. Damian, "Ecosystems in GitHub and a method for ecosystem identification using reference coupling," in *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR'15)*, 2015, pp. 202–207.

[112] L. Bauer, L. F. Cranor, R. W. Reeder, M. K. Reiter, and K. Vaniea, "Real life challenges in access-control management," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 899–908.

[113] R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, and M. Prabaker, "Field studies of computer system administrators: Analysis of system management tools and practices," in *Proceedings of the 2004 ACM conference on Computer Supported Cooperative Work*, 2004, pp. 388–395.

[114] D. Botta, R. Werlinger, A. Gagné, *et al.*, "Towards understanding IT security professionals and their tools," in *Proceedings of the 3rd Symposium on Usable Privacy and Security (SOUPS'07)*, 2007, pp. 100–111.

[115] M. Silic and A. Back, "Information security and open source dual use security software: Trust paradox," in *Open Source Software: Quality Verification*, E. Petrinja, G. Succi, N. El Ioini, and A. Sillitti, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 194–206.

[116] S. E. McGregor, P. Charters, T. Holliday, and F. Roesner, "Investigating the computer security practices and needs of journalists," in *Proceedings of the 24th USENIX Security Symposium (Sec'15)*, 2015, pp. 399–414.

[117] S. E. McGregor, E. A. Watkins, M. N. Al-Ameen, K. Caine, and F. Roesner, "When the weakest link is strong: Secure collaboration in the case of the Panama Papers," in *Proceedings of the 26th USENIX Security Symposium (Sec'17)*, 2017, pp. 505–522.

[118] C. Chen, N. Dell, and F. Roesner, "Computer security and privacy in the interactions between victim service providers and human trafficking survivors," in *Proceedings of the 28th USENIX Security Symposium (Sec'19)*, 2019, pp. 89–104.

[119] K. Gallagher, S. Patil, and N. Memon, "New me: Understanding expert and non-expert perceptions and usage of the Tor anonymity network," in *Proceedings of the 13th Symposium on Usable Privacy and Security (SOUPS'17)*, 2017, pp. 385–398.

[120] W. Bai, M. Namara, Y. Qian, P. G. Kelley, M. L. Mazurek, and D. Kim, "An inconvenient trust: User attitudes toward security and usability tradeoffs for key-directory encryption systems," in *Proceedings of the 12th Symposium on Usable Privacy and Security (SOUPS'16)*, 2016, pp. 113–130.

[121] M. Gutfleisch, J. H. Klemmer, N. Busch, Y. Acar, M. A. Sasse, and S. Fahl, "How does usable security (not) end up in software products? results from a qualitative interview study," in *Proceedings of the 43rd IEEE Symposium on Security and Privacy (S&P'22)*, 2022.

[122] K. Krombholz, W. Mayer, M. Schmiedecker, and E. Weippl, "I have no idea what I'm doing" - on the usability of deploying HTTPS," in *Proceedings of the 26th USENIX Security Symposium (Sec'17)*, Aug. 2017, pp. 1339–1356.

[123] K. R. Fulton, A. Chan, D. Votipka, M. Hicks, and M. L. Mazurek, "Benefits and drawbacks of adopting a secure programming language: Rust as a case study," in *Proceedings of the 17th Symposium on Usable Privacy and Security (SOUPS 2021)*, Aug. 2021, pp. 597–616.

[124] S. Roth, L. Gröber, M. Backes, K. Krombholz, and B. Stock, "12 angry developers - a qualitative study on developers' struggles with CSP," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS'21)*, 2021, pp. 3085–3103.

[125] L. Bieringer, K. Grosse, M. Backes, B. Biggio, and K. Krombholz, "Industrial practitioners' mental models of adversarial machine learning," in *Proceedings of the 18th Symposium on Usable Privacy and Security (SOUPS'22)*, Aug. 2022, pp. 97–116.

[126] J. Mink, H. Kaur, J. Schmüser, S. Fahl, and Y. Acar, ""Security is not my field, I'm a stats guy": A Qualitative Root Cause Analysis of Barriers to Adversarial Machine Learning Defenses in Industry," in *Proceedings of the 32nd USENIX Security Symposium*, 2023.

[127] D. Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, "Committed to trust: A qualitative study on security & trust in open source software projects," in *Proceedings of the 43rd IEEE Symposium on Security and Privacy (S&P'22)*, May 2022.

[128] D. Wermke, J. H. Klemmer, N. Wöhler, *et al.*, ""Always contribute back": A qualitative study on security challenges of the open source supply chain," in *Proceedings of the 44th IEEE Symposium on Security and Privacy (S&P'23)*, May 2023.

[129] https://reproducible-builds.org/.

[130] P. I. Fusch and L. R. Ness, "Are we there yet? data saturation in qualitative research," 2015.

[131] A. Offenwanger, A. J. Milligan, M. Chang, J. Bullard, and D. Yoon, "Diagnosing bias in the gender representation of HCI research participants: How it happens and where we are," in *Proceedings of the 2021 ACM Conference on Human Factors in Computing Systems (CHI'21)*, 2021.

[132] J. Jancar, M. Fourné, D. D. A. Braga, *et al.*, ""They're not that hard to mitigate": What cryptographic library developers think about timing attacks," in *Proceedings of the 43rd IEEE Symposium on Security and Privacy (S&P'22)*, 2022, pp. 632–649.

[133] https://reproducible-builds.org/specs/source-date-epoch/.

[134] K. Pettis and R. C. Hansen, "Profile guided code positioning," in *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation*, ser. PLDI '90, White Plains, New York, USA: Association for Computing Machinery, 1990, pp. 16–27.

[135] OpenSSF, *OpenSSF Scorecards*, https : / / securityscorecards.dev/.

[136] CISA, *Software Bill of Materials (SBOM)*, https://www.cisa.gov/sbom.

[137] *Executive order on improving the nation's cybersecurity*, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/, 2021.

[138] *Executive order on America's supply chains*, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/02/24/executive-order-on-americas-supply-chains/, 2022.

Appendix

For additional material, including the codebook, the questionnaire, and the motivational matrix, please visit https://publications.teamusec.de/2023-oakland-repro/.